MASTER'S THESIS

Master of Science in Advanced Mathematics and Mathematical Engineering

Title: Mathematical Optimization For Architectural Layout Design

Author: Moritz Otth

Advisor: Dr. Jordi Cortadella

Department: Department de Ciències de la Computatió (723)

Academic year:2020

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

Universitat Politècnica de Catalunya

Facultat de Matemàtiques i Estadística

Master in Advanced Mathematics and Mathematical Engineering

Master's thesis

Mathematical Optimization For Architectural Layout Design

Moritz Otth

Supervised by Dr. Jordi Cortadella

September, 2020

The gratitude should go to my advisor who gave me the possibilty to write the thesis of my master in a complex and interesting field.

As well I will thank my family for supporting me and always being there in difficult moments. Especially my brother in law, who always had time to listen, when I had to discuss my thoughts with somebody and to my dad who help with his opinion as well.

Finally my words of gratefulness go to all friends who had to cope with my different moods during the phases of progress and motivation, but nevertheless were so supportive.

Abstract

With the growth of population new living space is needed and new buildings, houses and flats have to be constructed. To construct one, is a time intensive process. An architect has to integrate different conditions and has to be beware of the restrictions. To find an optimal solution, sometimes is a complex challenge.

The approach of this thesis was, to write an algorithm which is handling exactly with this issue. If the algorithm is working, one could include constraints which are bounding the rectangles in moving or isolate them completely. Through that, one can safe resources in figuring out the different possibilities of how to manage the layout of a flat.

To allocate the rooms and to move them around, I wrote some algorithmic functions. The evaluation of the functions are done by IPOPT, a non-linear optimizer. I will describe them and show how they are working.

One can see which approaches were satisfying and which ones ended up without any success. The challenges, which I were faced with, are pointed out and I will describe how they can be solved.

Keywords

optimization, digital architect, architectural layout, programming, IPOPT

Contents

1. Introduction

1.1 Motivation and objective

Tetris is a well known game all over the world. This is a game, where you have different geometric figures and you have to place them correctly to save as much space as you can. This skill of optimisation is often useful in real life, too. If a carpenter has to slice a wooden plate in the way that he has the least amount of wasted wood, if a logistic employee has to pack a box with as many items as possible in the way that they use as few as possible boxes or if an informatician has to place chips on the motherbord of the computer, such that they can minimize the lengths of the trails connecting the elements.

The technical development increased in the last years enormously and with it the capacity of computers, calculation machines, servers etc. In the past, situations like the ones above were faster calculated by humans brain and the risk of mistakes was smaller than the time and the resources which computers needed. Over the years and with the increasing possibilities and capacity of artificial intelligence, engineers started to produce software which is caring exactly about these issues.

In the recent years a lot of scientist started to investigate several issues in that context.
The article [1] takes care about packing rectangles in a rectangular container with the aim to waste as little space as possible. The way of packing is simple, it takes one rectangle after the other and tries to place them as close as possible to the previous ones or to the border of the container as shown in Figure 1. These rectangles are independent and do not have any relation to each other. So, after calculating the loss of space with all the different combinations of the orders of the rectangles, the optimal solution would be found.

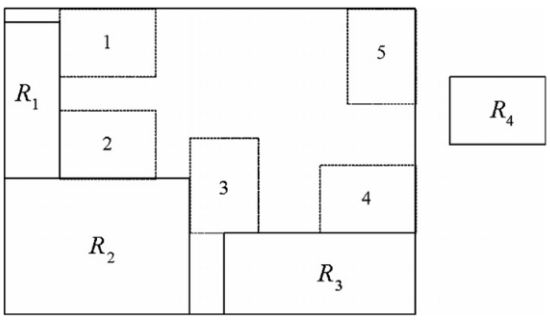


Figure 1: 2D rectangular packing problem: The rectangles are labeled with R_i, $i = 1, \cdots, 4$ and the possible positions with numbers j, $j = 1, \cdots, 5$

In the article [2] the problem is already a bit more complex. It is trying to fit figures of different shapes in a box, again with the goal to waste as little space as possible. As in the article [1] the different figures are independent, so there are no constraints of how they have to be ordered.

In the article [3] they are investigating the rectangle blanket problem. The rectangle blanket problem cares about putting a blanket over an image of nonoverlapping axis-alinged rectangles with the target to make the blanket as small as possible. The blanket and the rectangles do not have to cover them fully, but the subject is to have an upper bound of how many rectangles fit in the blanket with the least loss of space. But still, the rectangles are not bound to conditions.
This topic already comes closer to the content of the thesis.

This thesis is a continuation of the work about distributing computer chips on the mother board. The work of the distribution of computer chips is concerning as well the divison of nonoverlapping rectangles, but with the goal that the lengths of the cables between the connected computer chips is minimized.

The idea of this thesis arised out of the problem of the computer chips. The purpose is to establish an algorithm which is calculating the optimal distribution of rooms in a flat under several circumstances. For example, they all have to be connected with a corridor or all the rooms which need water access are placed next to each other. The program should enable the operator to insert the size of the flat, the number of rooms, the limits of what is possible or not and his imaginations in form of constraints.

We assume that all the rooms are rectangular, like the flat. So all the objects have the same geometric shape, but the sizes - length and height - are different.

The thesis is an interaction between mathematics, computer science and architecture. The topic is based in computational geometry. Computational geometry is a part of computer science which deals with the study of algorithms with geometrical basis. Because these issues are not linear, IPOPT, (Interior Point OPTimizer) a nonlinear optimization software, is used to do the calculations. A big advantage of IPOPT is that there are a lot of options which can be included.

In a first step the aim was to create an algorithm which is moving the rectangles until they where axis-alinged but not overlapping. After this basic step is done, one can proceed with the different conditions of the distribution of the flats.

The final positions of the rooms depend on the initial points. Because of that it is crucial that one takes the optimal starting point. This is a complex calculation which is treated in the architectual layout problem.

1.2 Report organization

The report is organized as follows. After the introduction, we will explain in chapter 2 (Background) the concept of the thesis, the way how we define our variables and how we are formulating a flat and its borders. In the same chapter I will explain IPOPT and the gradient descent method, too. This helps to understand the theoretical background.

In chapter 3 (Algorithms) I am describing our different algorithms which I wrote. I listed the codes in the text and described what they are doing. I will explain the overlap, the gradient and the distance function. At the end of this chapter I will as well present how I am doing the visualization.

In the chapter 4 (Results and discussion) I am reconsidering about how I proceeded and what the achievements are. I am describing the process from the beginning, which attempts were successfully and which not and what changes I had to do, over were I got stuck till with what result I ended up.

In the last chapter (Conclusion) I will conclude my work. I will reconsider the hypothesis of the project, what I achieved and with what I had to finish. After that I will give a statement to the personal evaluation and development. To finish the chapter and the paper I will look on future works, what could be done to continue with that topic.

2. Background

In this section I will introduce the concept of how we obtain a flat and its rooms, a short summary how IPOPT is working and the gradient descent method which IPOPT uses to do the calculation.

2.1 Concept

We are trying to calculate the best distribution of rooms in a flat under specific conditions. The flat is a restricted area where the rooms are placed in and where they can be moved independently as long as they do not violate the constraints. This area is equal to the shape and the square meters of the flat. We will imagine the area as a coordinate system, where the lower left corner is placed on the origin, the width of the flat will be the upper bound for the x-coordinates and the height of the flat the upper bound for the y-coordinates.
The rooms will be interpreted as rectangles which we will float freely to different positions in the area of the flat. Every room is initalized with four points, two for the left corner, one for the width and one for the height:

- x_i: x-coordinate for the lower left corner.
- w_i: width of the room, which will be the x-coordinate of the lower right corner of the rectangle.
- y_i: y-coordinate for the lower left corner.
- h_i: height of the room, which will be the y-coordinate of the upper left corner of the rectangle.

Definition 2.1 *A room is a rectangle defined by the lower left corner* (x_i, y_i)*, its width* (w_i) *and its height* (h_i)*. The rectangles are written in the array x, where the coordinates of the rectangles are listed after each other in the following way,* $x = \{x_1, w_1, y_1, h_1, x_2, w_2, y_2, h_2, \cdots\}$.

When implementing the rectangles it can occur that they are overlapping. Depending on the initialized points, the order and the final positions of the rectangles can vary. This happens because we are using the gradient descent method which always takes the fastest descent and sometimes ends up in a local minimum, but we will explain the method later in the paper. Through IPOPT we will minimize the overlapping area with regard to the rectangles. The algorithms are explained in the next chapter.
We will implement as well some constraints which will give IPOPT a frame for how it can change position and the shape of the rectangles. On the one hand, the constraints are formulating the restrictions, such as the area of the flat and on the other hand it is possible to include the specifications of the operator or the client. An example would be the shape of the rooms, such that they not end

up with having the shape of fries. Another constraint would be, that some of the rooms, for example the living room and the bed rooms have to be at the outside wall of the flat, so it is possible to place a window in these rooms.

2.2 IPOPT

IPOPT (Interior Point Optimizer) is a software which is used for large-scale nonlinear optimizations. It solves nonlinear programming problems of the form:

$$\begin{aligned} & \min_{x \in \mathbb{R}^n} && f(x) \\ s.t. \quad & && g^L \leq g(x) \leq g^U \\ & && x^L \leq x \leq x^U, \end{aligned} \tag{2}$$

where $x \in \mathbb{R}^n$ are the variables which have to be optimized. $g(x)$ discribes the constraints of the function, where we can set a lower bound $g^L(x)$ and and upper bound $g^U(x)$ for them. The variables $x[i]$ can be restricted too. x_i^L is the lower bound and x_i^U is the upper bound for the variable x_i. [1]

IPOPT tries to minimize the objective function using the gradient descent method. For that it needs the objective function, its gradient, some constraints, the Jacobian of the constraints and the Hessian of the Lagragian. The Hessian can as well be calculated with a quasi-Netwon approximation which we will use in this thesis. Instead of a mathematical function we will use algorithmic ones which will be described later herein.

To compile the software, a code file is needed, a NLP file to interface with IPOPT, a compiler file and an execute file. If one will implement different options, there is the option to implement them or to refer to other files where IPOPT gets the information for the specific option. We will give a quick overview how it looks like, but neglect intentiously several details. The files as the functions are to be found in more details in the official documentation of IPOPT.

In the compiler file are the default settings, where we can define additional instructions. Some of the settings are necessary and cannot be changed once the code is running. With the additional options, we can add settings like the usage of the quasi-Newton approximation for the Hessian.

In the NLP file, we specify the different functions which will be used in the code.

[1] Andreas Wächter, IPOPT documentation, 2002, https://coin-or.github.io/Ipopt/

To run the code we need to provide some information to IPOPT. We will list those we need for this thesis.

- Problem dimension
  - number of variables
  - number of constraints
- Problem bounds
  - variable bounds
  - constraint bounds
- Initial starting point
  - Inital values for the primal x variables
- Problem Structur
  - number of nonzeros in the Jacobian of the constraints
  - sparsity structure of the Jacobian of the constraints
- Evaluation of Problem Functions
  Information evaluated using a given point (x coming from IPOPT)
  - Objective function, $f(x)$
  - Gradient of the objective, $\nabla f(x)$
  - Constraint function values, $g(x)$
  - Jacobian of the constraints, $\nabla g(x)^T$

As already described above, every rectangle has four coordinates, so we will have the number of rectangles n times 4 variables. Some of the constraints will descripe the shape of each room. Additionally we have further constraints which are keeping the directions of the rectangle, so that the coordinates for the width are always on the right side and those for the height are always higher than those of the left corner.

2.3 Gradient descent method

2.3.1 Introduction example

To introduce the gradient descent method we start with an analogical example. Imagine a person is on a mountain and wants to go down as fast as possible. This person can just see as far as its

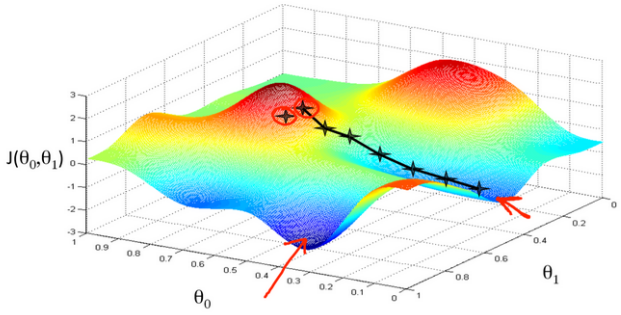


Figure 2: Example of the gradient descent method with two different inital points.

next step, respectively doesn't see the valley at all. At each step he/she has to decide again in which direction the steepest descent is, before making the next step.
With this condition the person decides to take the fastest descent till he/she will be at a local or the global minimum. But he/she does not know if he/she will reach the global minimum (the valley) or not. For example if the person steps down and arrives at a mountain lake, he/she is at a local minimum, because it will not go further down. Either he/she is on the same level as the lake or the person has to ascend again, but walking downwards is not possible. In this case he/she would not have reached the target where he/she wanted to end.
So the end position of the person is depending on the starting point. Different starting points will bring different ways to step down. As we can see in figure 2 there are different ways of stepping down with different outcomes. The descent from the right starting point (top point) is marked with the black crosses and lines. But if we would take the left starting point, we would end up at the position which is marked with the left red arrow.

2.3.2 Mathematical background

Given a multi-variable function $F(x)$ which is differentiable in a neighbourhood of a point a. The gradient descend method takes the negative gradient of $F(x)$ adding to the previous point and is building the recursive sequence of x.
We have that

$$a_{n+1} = a_n - \alpha \nabla F(a_n) \tag{3}$$

for $\alpha \in \mathbb{R} \geq 0$. Following this sequence we want to achieve the local minimum. So with an arbitrarily starting point x_0 such that

$$x_{n+1} = x_n - \alpha \nabla F(x_n), \quad n \geq 0 \tag{4}$$

we get the monotonic sequence

$$F(x_0) \geq F(x_1) \geq F(x_2) \geq \cdots \tag{5}$$

which arrives at a local minimum.[2]

[2]Wikipedia contributors, Gradient descent, 2020, https://en.wikipedia.org/w/index.php?title=Gradient_descent&oldid=962649577

3. Algorithms

In this section I will describe the algorithms I made and show how they are working. The algorithms which are shown below, are exported from the IPOPT file, so that they could run independently for our example.

In figure 3 we see the example which we use to describe the functions, $R_1 = \{x_1, w_1, y_1, h_1\} = \{0, 6, 0, 9\}$ and $R_2 = \{x_2, w_2, y_2, h_2\} = \{3, 9, 4, 11\}$. The rectangles are saved in an array one after the other, $x[] = \{R_1, R_2\} = \{0, 6, 0, 9, 3, 9, 4, 11\}$. We are counting the overlap by the amount of squares where they do overlap. So in here they have an overlap of 15. For IPOPT we need the overlap function as the cost function which we try to minimize and the gradient function to calculate how the overlap will differ if we change one of the four coordinates of the rectangle. If the gradient is zero, but they are overlapping, we will increase the distance between the centers of the rectangles to push them apart.

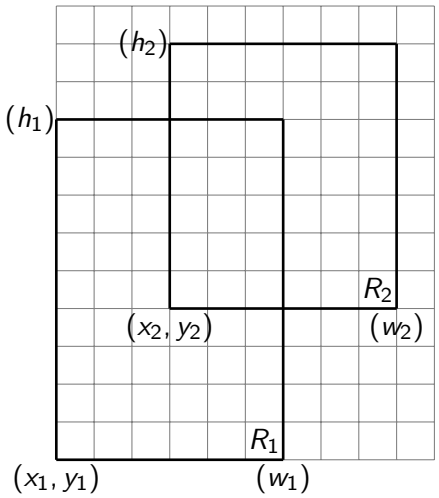


Figure 3: Example of two rectangles with an overlap of 15. The corners are marked with the values we use in the function to define the rectangles.

3.1 Overlap

Definition 3.1 The Overlap function calculates the total amount of overlapping area for all combinations of two rectangles. In other words, we will have

$$\textit{Total overlap} = \sum_{i,j \text{ for } i<j} \textit{Overlap}_{i,j} \tag{6}$$

which gives us $\binom{n}{2}$ combinations of rectangles.

In this function we are iterating over two loops, the first one is for indicating the first rectangle and the second one for all the other rectangles which we haven't counted the overlap with the first one left. The *if* $(i < j)$-loop controlls that we are not counting a combination twice. If we would not have it, for example we would count the overlap for R_1 to R_2 and afterwards the overlap for R_2 to R_1.

In the main part of the function we are taking the bigger of the two x-coordinates as the minimum value *xmin*. For the maximum values we are taking the smaller of the widths w_i of the two rectangles. If we are looking at our example, $xmin = x_2 = 3$ and $xmax = w_1 = 6$. Subtracting *xmin* from *xmax* we are getting the overlap for the x-coordinate. After that we are doing the same for the y-coordinate and the height, so $ymin = y_2 = 4$ and $ymax = h_1 = 9$. If the overlaps for both coordinates are calculated, we are multiplying the two numbers and get the overlap of these two rectangles. At the end of each iteration we are adding this number to the local variable *overlap_area*.

Finally after all iterations we are returning the total amount of overlap (*overlap_area)* back to the objective value for IPOPT.

```
// Computes the amount of overlap between two rectangles
double Overlap (double x[], int n1){
    double overlap_area = 0;
    for (int i = 0; i < n1; i++)
    {
        for (int j = 0; j < n1; j++)
        {
            if (i < j)
            {
                // overlap in the x dimension
                double xmin = max(x[i*4], x[j*4]);
                double xmax = min(x[i*4+1], x[j*4+1]);

                // overlap in the y dimension
                double ymin = max(x[i*4+2], x[j*4+2]);
                double ymax = min(x[i*4+3], x[j*4+3]);
```

```
                if ((xmax-xmin)*(ymax-ymin) > 0)
                overlap_area = overlap_area + (xmax-xmin)*(ymax-ymin);

            }
        }
    }

    return overlap_area;
}

int main() {
    // Coordinates of the rectangles
    // x: x-coordinate, y: y-coordinate, w: weidth, h: height -
    // (w,h: distance from left/botton to right/top side of the rectangle)
    double x[] = {0,6,0,9,3,9,4,11,7,13,2,7};
    int n1 = sizeof(x)/32;

    double obj_value = 0;
    obj_value = obj_value + Overlap(x,n1);
}
```

3.2 Gradient of the overlap

Definition 3.2 The gradient of the overlap is calculated algorithmically. It is defining the modification which occurs by changing one of the coordinates of a rectangle.

In Chapter 2.3 we already explained the gradient descent method. If we are applying that on the movements of rectangles, we are calculating the direction in which we can minimize the overlap with the biggest amount.
Applying this on our example we will get the following gradient for the first application.

$$
\begin{aligned}
O &= \mathit{overlap} \\
\frac{\partial O}{\partial x_0} &= 0; \qquad \frac{\partial O}{\partial x_4} = -5; \\
\frac{\partial O}{\partial x_1} &= 5; \qquad \frac{\partial O}{\partial x_5} = 0; \\
\frac{\partial O}{\partial x_2} &= 0; \qquad \frac{\partial O}{\partial x_6} = -3; \\
\frac{\partial O}{\partial x_3} &= 3; \qquad \frac{\partial O}{\partial x_7} = 0;
\end{aligned}
\tag{7}
$$

As we can see, if we would increase the width of the first rectangle by one, the amount of overlap

would increase by five. The same counts for the height of R_1. By changing the x-coordinate of R_2 the overlap reduces by 5 and for the y-coordinate by 3. Changing the other coordinates does not have an impact in this case. But if there are more than two rectangles, it could be that one of them increases or decreases the overlap.

As in the previous function we will iterate again over all the rectangles, but here we will calculate the gradient for every rectangle comparing with the others, for that we have the condition ($i! = j$).
Before the start of the function itself, we are producing some free space to save the array *Grad* with the malloc function.
As in the overlap function the gradient function calculates again the x- and the y-overlap because the gradient is changing along the overlapping distance of each coordinate. So if the y-overlap is 5, as in R_1 in our example, a modification of the width of R_1 brings an increase of 5. If the overlapping area of two rectangles is less than zero, the gradient will be zero too, because they are either overlapping nor touching each other. Otherwise at least one position of the gradient will be bigger than zero. If so we are checking for which coordinate the gradient will be different to zero. Every coordinate with a value different from zero will be placed at the corresponding position in the *Grad* array. So if a rectangle is overlapping or touching with multiple other rectangles, the modification will be saved as a total.
We will returning *Grad*, so that we can use the single positions of it for the evaluation of the gradient of the objective function from IPOPT. If IPOPT has evaluated the gradient of the objective function, we will free the space which we reserved with malloc, so the space can be used in the next iteration again.
In our example we will get back the following array for the gradient:

$$Grad[] = \{0, 5, 0, 3, -5, 0, -3, 0\}. \tag{8}$$

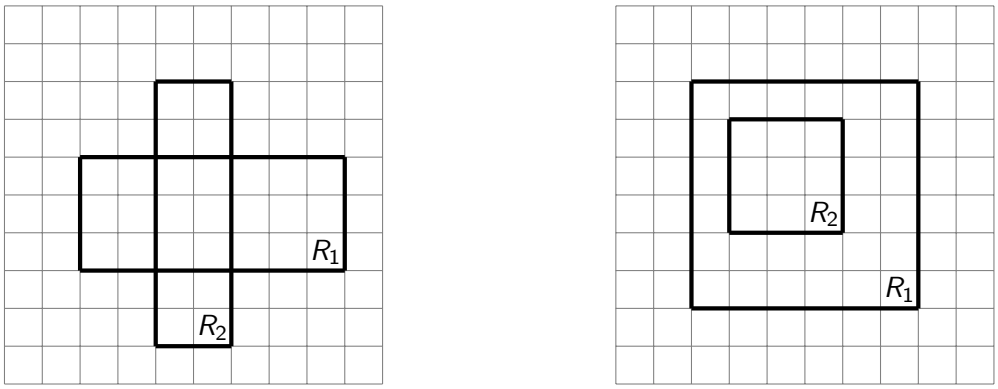
```
// Calculates the gradient of the variablhugees of the rectangles with regard
// to the overlap with the other rectangles. The gradient is returned as a result
// of the functionhuge

double* Gradient(double x[], const int n1) {
        // We are calculating the overlap of the rectangles. If the apart of each
        //  other, the gradient is zero. If they are overlapping or touching, the
        // gradient is different to zero at at least one position.
        double *Grad = (double *)malloc( n1 * 4 * sizeof(double) );

    for (int i = 0; i < n1; i++)
    {
        for (int j = 0; j < n1; j++)
        {
```

```
        if (i != j)
        {
            double xmin = max(x[i*4], x[j*4]);
            double xmax = min(x[i*4+1], x[j*4+1]);

            double ymin = max(x[i*4+2], x[j*4+2]);
            double ymax = min(x[i*4+3], x[j*4+3]);

            double x_overlap = xmax-xmin;
            double y_overlap = ymax-ymin;

            if (x_overlap < 0)  x_overlap = 0;
            if (y_overlap < 0)  y_overlap = 0;

            if (x_overlap * y_overlap >= 0)
            {
                // Gradient for xmin
                if (x[i*4] >= x[j*4] && x[i*4] != x[j*4+1] && x[i*4+1]
                != x[j*4] && x[i*4+2] != x[j*4+3] && x[i*4+3] != x[j*4+2])
                Grad[i*4] += -y_overlap;

                // Gradient for xmax
                if (x[i*4+1] < x[j*4+1] && x[i*4+1] != x[j*4] && x[i*4]
                != x[j*4+1]) Grad[i*4+1] = Grad[i*4+1] + y_overlap;

                // Gradient for ymin
                if (x[i*4+2] >= x[j*4+2] && x[i*4] != x[j*4+1] && x[i*4+1]
                != x[j*4] && x[i*4+2] != x[j*4+3] && x[i*4+3] != x[j*4+2])
                Grad[i*4+2] = Grad[i*4+2]-x_overlap;

                // Gradient for ymax
                if (x[i*4+3] <= x[j*4+3] && x[i*4+3] != x[j*4+2] && x[i*4+2]
                != x[j*4+3]) Grad[i*4+3] = Grad[i*4+3] + x_overlap;

                // Gradient for xmax if rectangles are touching by
                // the x-coordinate
                if (x[i*4+1] == x[j*4] && x[i*4+3] > x[j*4+2]
                && x[i*4+2] < x[j*4+3])
                {
                    Grad[i*4+1] = Grad[i*4+1] + y_overlap;
                }

                // Gradient for ymax if rectangles are touching by
                //the y-coordinate
                if (x[i*4+3] == x[j*4+2] && x[i*4+1] > x[j*4]
                && x[i*4] < x[j*4+1])
                {
```

```
                        Grad[i*4+3] = Grad[i*4+3] + x_overlap;
                    }
                }
            }
        }
    }
    return Grad;
}

int main() {
    // Coordinates of the rectangles
    // x: x-coordinate, y: y-coordinate, w: weidth, h: height -
    // (w,h: distance from left/botton to right/top side of the rectangle)
    double x[] = {0,6,0,9,3,9,4,11,7,13,2,7};
    int n1 = sizeof(x)/32;

    double *grad = NULL;
    grad = Gradient(x, n1);

    free(grad);
}
```

3.3 Distance

If we are calculating the gradient of the overlap it could happen, that we get a gradient of all zeros. That could be, because they are not overlapping at all or because the cost of the overlap is locally constant which means that one rectangle is included in the other one or it is crossing through the whole rectangle as in figure 4 shown.



(a) Crossing of R_1 and R_2. (b) R_1 includes R_2.

Figure 4: Example for locally constant cost of overlap.

To avoid that these situations appear, we are including the distance function.

After calculating the gradient, we are checking through a loop if the gradient of each rectangle is zero or not. The distance functions is only used, when the gradient for each variable of a rectangle is zero. If one position is different to zero, then there is a direction where it can move to in the next step. But if there is no direction, the algorithm would get stuck. So if the gradient is zero, we are calling the distance function.

The distance function is always just call for one rectangle, namely the one we checked has a gradient of all zeros. At first in the function we are checking, if the rectangle is locally constant with another one or if it is not overlapping with anyone at all. If they do not overlap, the function finishes without changing anything. Its algorithm will go back to the loop and checks if there is another rectangle with gradient equal to zero.
If the rectangle and another one are locally constant, the function continues. It then takes the center of the two rectangles and looks what are the positions towards the other one and gives it a number. So if the rectangle is included or crossing another one and the center of it is more to the right and more to the top of the one which is including or crossing it, then it will get a one.

Number	Position
1	top right
2	bottom right
3	bottom left
4	top left

Then the function is checking if the rectangle is locally constant with another rectangle. If there is a further one, it will add the number to the already given one. If all the combinations are checked, the function returns the total number.
If the total number is zero, we know, that there is no rectangle which is overlapping with this one. In all the other cases, we take the number modulo 4. The number which is resulting, shows us in which direction the rectangle should move and because of that it is changing the gradient of that rectangle, so that it is not all zeros anymore.

```
for (int i = 0; i < n1; i++)
{
    if (grad_1[i*4] == 0 && grad_1[i*4+1] == 0 &&
            grad_1[i*4+2] == 0 && grad_1[i*4+3] == 0)
    {
        dist1 = Distance(x,n1,i);
        dist = dist1[0];

        if (dist == 0){goto finished;}

        dist = fmod(dist,4);
```

```
            // if dist == 1 the rectangle should move to the top right
            if (dist == 1)
            {
                grad[i*4] = grad[i*4] + 1;
                grad[i*4+1] = grad[i*4+1] + 1;
                grad[i*4+2] = grad[i*4+2] + 1;
                grad[i*4+3] = grad[i*4+3] + 1;
            }
            // if dist == 2 the rectangle should move to the bottom right
            if (dist == 2)
            {
                grad[i*4] = grad[i*4] + 1;
                grad[i*4+1] = grad[i*4+1] + 1;
                grad[i*4+2] = grad[i*4+2] - 1;
                grad[i*4+3] = grad[i*4+3] - 1;
            }
            // if dist == 3 the rectangle should move to the bottom left
            if (dist == 3)
            {
                grad[i*4] = grad[i*4] - 1;
                grad[i*4+1] = grad[i*4+1] - 1;
                grad[i*4+2] = grad[i*4+2] - 1;
                grad[i*4+3] = grad[i*4+3] - 1;
            }
            // if dist == 0 the rectangle should move to the top left
            if (dist == 0)
            {
                grad[i*4] = grad[i*4] - 1;
                grad[i*4+1] = grad[i*4+1] - 1;
                grad[i*4+2] = grad[i*4+2] + 1;
                grad[i*4+3] = grad[i*4+3] + 1;
            }

            finished: cout << "dist = 0." << endl;

        }
    }
```

```
double* Distance (const double x[], const int n1, const int i){
    double* change = 0;
    for (int j = 0; j < n1; j++)
    {
        if (i != j)
        {
            double grad_change = 0;
            // R2 includes R1
```

```
        if (x[i*4] > x[j*4] && x[i*4+1] < x[j*4+1] &&
            x[i*4+2] > x[j*4+2] && x[i*4+3] < x[j*4+3])
        {
            grad_change = 1;
        }
        // R1 includes R2
        if (x[i*4] < x[j*4] && x[i*4+1] > x[j*4+1] &&
            x[i*4+2] < x[j*4+2] && x[i*4+3] > x[j*4+3])
        {
            grad_change = 1;
        }
        // R1 is crossing R2 such that R1 is larger in the vertical axis
        if (x[i*4] > x[j*4] && x[i*4+1] < x[j*4+1] &&
            x[i*4+2] < x[j*4+2] && x[i*4+3] > x[j*4+3])
        {
            grad_change = 1;
        }
        // R1 is crossing R2 such that R1 is larger in the horizontal axis
        if (x[i*4] < x[j*4] && x[i*4+1] > x[j*4+1] &&
            x[i*4+2] > x[j*4+2] && x[i*4+3] < x[j*4+3])
        {
            grad_change = 1;
        }
        // investigate where the centers of the rectangles are.
        // 1 = top right; 2 = bottom right; 3 = bottom left; 4 = top right.
        if (grad_change == 1)
        {
            double center_x1 = 0;
            double center_y1 = 0;
            double center_x2 = 0;
            double center_y2 = 0;

            center_x1 = x[i*4+(i*4+1)/2];
            center_y1 = x[i*4+2+(i*4+3)/2];
            center_x2 = x[j*4+(j*4+1)/2];
            center_y2 = x[j*4+2+(j*4+3)/2];

            if (center_x1 > center_x2 && center_y1 > center_y2)
                    {change = change+1;}
            if (center_x1 > center_x2 && center_y1 < center_y2)
                    {change = change+2;}
            if (center_x1 < center_x2 && center_y1 < center_y2)
                    {change = change+3;}
            if (center_x1 < center_x2 && center_y1 > center_y2)
                    {change = change+4;}
        }
    }
```

```
    }
    return change;
}
```

3.4 Visualization

IPOPT returns the result of the calculation in raw numbers which is complex to evaluate, if there are a lot of rectangles. For that reason we are visualizing the data in a gif, which will be compiled with a python code. From IPOPT it is possible to save the final data in a text file, such that we just can read this file with python and using the information to produce the gif.

The first step is to read the file which we have from IPOPT and save the data in the list *rect*. After that we will call the function to produce the images. For this we need the width and the height of the image, the coordinates of the rectangles and the number of rectangles. The function then will create an image with all the rectangles in it. We color them differently, so that it is easier to differ. In the end we will add the images to a sequence of images to produce the gif from it. We can controll the speed of the gif, so if there are a lot of images, it is possible to speed it up, to arrive the final state of the rectangles.

```
from PIL import Image, ImageDraw
rect = []
f = open("final_coordinates_rectangles.txt", "r")
f = f.read()
x = f.split(",")
x.remove(x[-1])
for i in range(len(x)):
    rect.append(float(x[i]))

# empty list for saveing the rectangles.
rectangles = []

# Function to visualize the overlapping rectangles.
def overlap_visualization(width, height, rectangles,number_of_rectangles):
    img = Image.new('RGB', (width, height), (255, 255, 255))
    draw = ImageDraw.Draw(img)
#    List of colors for the different rectangles -> will change to automatical
#    colors, so that the index number i gives the color of the rectangle.
    color = ["red", "blue", "green", "yellow", "gray", "black", "lila", "pink"]

#    Producing the rectangles and draw them.
    for i in range(number_of_rectangles):
        shape = [(rectangles[i*4]*5,250-rectangles[i*4+2]*5),
                 ((rectangles[i*4] + rectangles[i*4+1])*5,
                  250-(rectangles[i*4+2] + rectangles[i*4+3])*5)]
```

```
        draw.rectangle(shape, outline = color[i])

    return img

rectangles.append(rect)

# Calculates the number of rectangles in the picture.
number_of_rectangles = int(len(rectangles[0])/4)

# Create the frames.
frames = []

# Calculating the steps of moving, so how many iterations till the rectangles
# do not overlap anymore.
len_rectangles = len(rectangles)

# Calling the function and adding the images to the frame, so we can produce
# the gif with the frame.
for i in range(len_rectangles):
#    the first two numbers are for the size of the background, the third number
#        is for the step of the rectangles, and the last number for the number
#        of rectangles.
    new_frame = overlap_visualization(250,250,rectangles[i],number_of_rectangles)
    frames.append(new_frame)

# Save into a GIF file.
frames[0].save('rectangles.gif', format='GIF', append_images=frames[1:],
      save_all=True, duration=2000) # loop = 0 if it should loop forever.
#    duration gives the speed of moving the rectangles.
```

4. Results and discussion

The beginning phase of the project was about to get familiar with the idea itself. So I started to read different articles about packing problems. After a few weeks reading throught articles I could conclude that there was no constellation similar to this one. Most of the articles were about packing rectangles in a certain order, placing them one by one. Some care about rectangles and moving them apart, but often there was enough space available, so they didn't have a limited area such as in a given flat.

In the second step I started writing the code for the algorithm. First I had to define the function which is calculating the overlap of the rectangles. Because this is a well-known issue, a lot of reports gave many different suggestions. At the beginning, I started with saving the variables in an array, but because this is not a proper way to store variables, I switched it to vectors. The vectors were still difficult to read, so I switched them to structs. To change from vectors to structs I had to rewrite almost the whole code, which was very timeconsuming. After rewriting the function to structs I started with the next step and tried to include it in IPOPT. The example in IPOPT was using arrays as well, so I tried to change it, enabling me to use my code with the structs. After a while I realized that changing the body of IPOPT was more difficult than changing the body of my function. This meant that I had to rewrite the whole code again back to arrays. Continuing with arrays I could implement the code.

The objective function was ready, so the next step was to write an algorithm which is calculating the gradient of the objective function. To write a function in an indepentend environment was fastly done. It worked as expected. By implementing the gradient function to IPOPT the first problems occured. By running the algorithm, the program always collapsed and it only showed "Segmentation fault (core dumped)". It was a problem of saving the variables, so I had to switch the function to a pointer function. Additionally I had to include some external saving space with the malloc function so that the function could save the variables temporarly to reengage them outside of it.
IPOPT calculated float gradients and as well negative ones. As well it run over different if-loops, so the gradient was dobble the size as it should have been. Because of that the function did not work properly. I figured out that the problem was by calculating the overlapping area. The overlap function did not only calculate overlapping area, it calculated negative overlapping areas, too. This is not possible. So I had to include an option for selcting if one of the calculated areas was negative. A negative area means that they are not overlapping. Running the algorithm again, the result weren't still not correct. It turned out that creating saving space with malloc, the saving space did already have some variables placed in, so clearly the results couldn't fit. The saving space had to be set to zero.

The code was running so I wrote another code to visualize the results, therefore it was easier to

retrace what IPOPT was calculating. The rectangles were still not overlapping and the algorithm stopped before they weren't. Starting with other initial points did help either, there always left an overlapping area as in figure 5 shown.

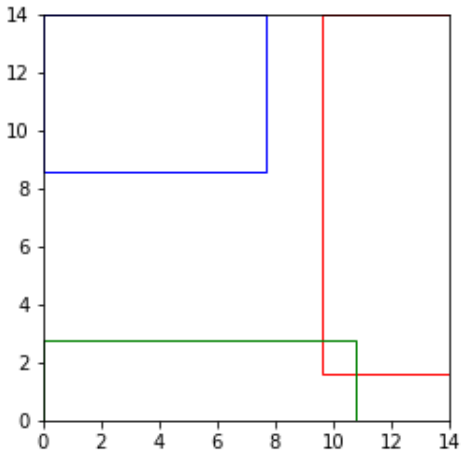


Figure 5: Plot of the results after a calculation, where the rectangles are still overlapping.

Thanks to my advisor, the next idea was to push the centers away from each other. I tried to include a function in the objective function, which is measuring the distance of one rectangle to the others and tries to push them apart. Because IPOPT tries to minimize the objective function, pushing the rectangles apart of each other just increased the objective value. The algorithm then stopped in a mix between pushing them apart and minimizing the overlapping area. This was not the objective and I set some factors in front of the overlapping area function and the function for pushing them apart. The goal was to weight first one of the functions higher and to reduce the weight during the time, while the other function was weighted less and was growing with the time.

$$obj\_value = \alpha \sum_{\forall i,j} (D_i - d_j)^2 + \beta \sum_{\forall i,j} Overlap_{ij} \tag{9}$$

The first part of the equation was taking the distance between to rectangles, the second part the overlapping area. The distance was measured by the left bottom points of the rectangles.
I tried to put the factors in front of both functions, so once with the smaller starting factor in front of the distance function and once in front of the overlap function. Neither of both came to a satisfying result.

It turned out that pushing them apart was only necessary, when the gradient was zero. As discribed in section 3.3 a gradient of zero could be if they are not overlapping nor touching or if they were

locally constant. So I excluded the function and wrote a new one, the one which is described in section 3.3. If a gradient is locally constant, this function takes the center of the rectangles and tries to increase the distance between the centers, so that by time one rectangle would not be included or crossing another one anymore. The problem by taking the left corners was that it did not matter what the position of the rectangles was, it always tried to push them to the top right corner if they were included in each other (see figure 6) or to the bottom right corner if they were crossing.

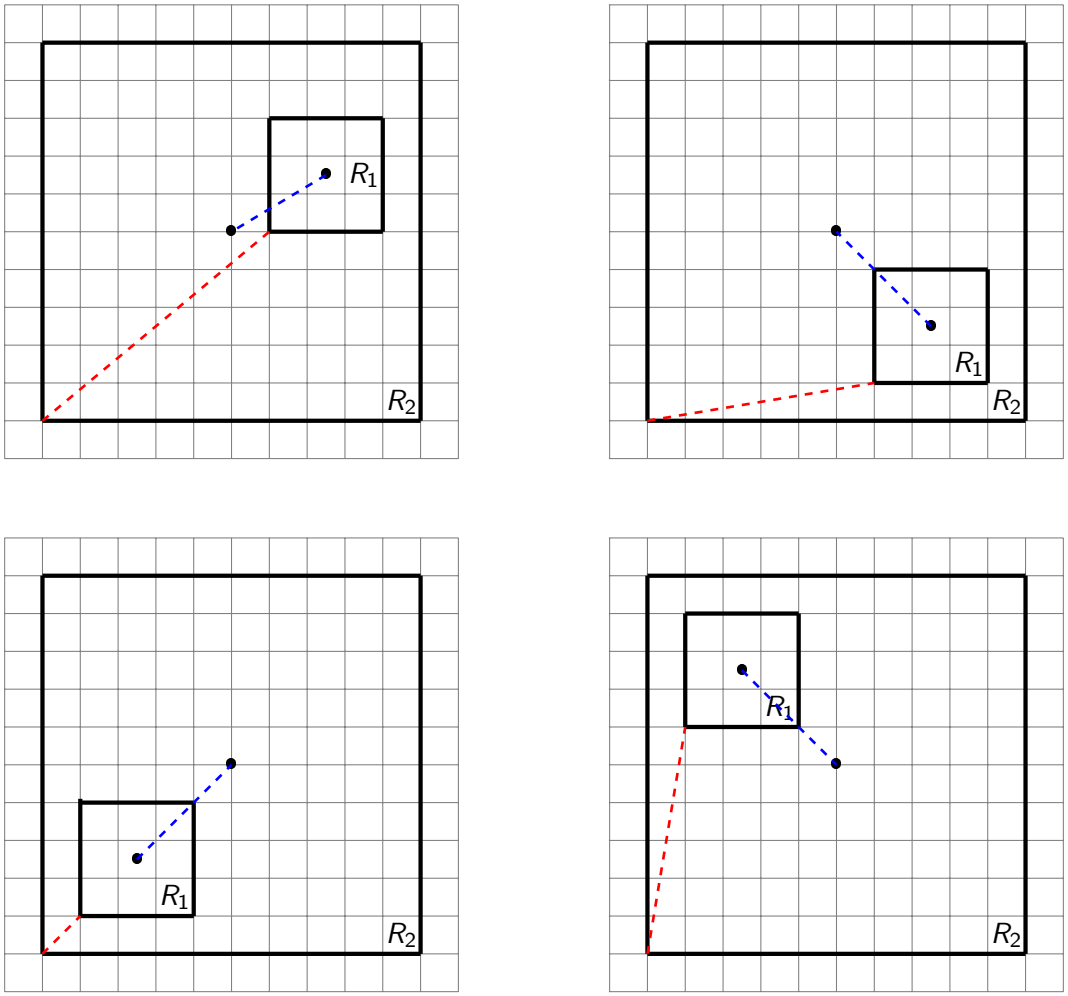


Figure 6: Example for rectangle R2 includes rectangle R1 in the four different positions. If we take the center for measuring the distance, we see that the blue line is always pointing in different direcctions. If we would take the left corner of each rectangle, the red line is always going to the top right, just in different angles.

The problem if it pushes all the rectangles in the same direction is that if more than two rectangles are locally constant, it pushes all for example to the top right corner and at the end, few of the rectangles getting stuck in this corner and cannot move anymore, because there is no more chance to get more to the top right.

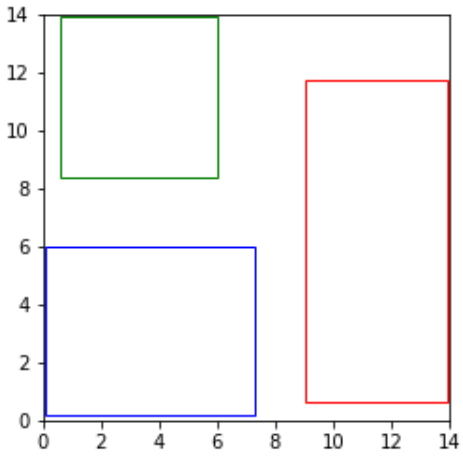


Figure 7: Plot of the results after a calculation with the distance function when the rectangles are not overlapping anymore.

Increase the distance of the centers help to bring them apart. Still, it does not work in all cases. Depending on the initial values it sometimes ends up with no overlapping area, like it is shown in figure 7. But in other cases the function achieves the opposite of we wanted. In figure 8 we see an example of how it looks if the function reaches the wrong result.

After several tests with different initial values I was wondering, what the difference is of using the distance function or not using it. So I did a few runs with the function and a few without the function. It turned out that the efficiency of both variation is almost equal. So I can conclude that the distance function does not have a strong impact on the results and running with only the overlapping function brings similar results (see figure 9).

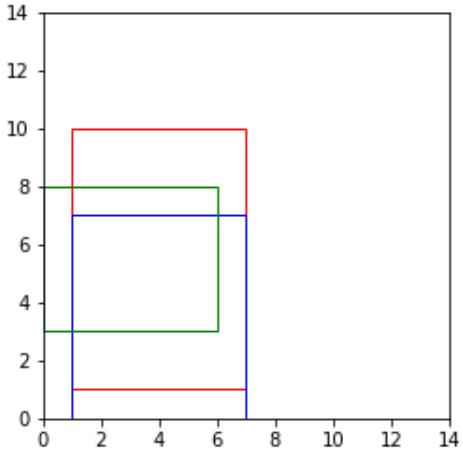


Figure 8: Plot of the results after a calculation with the distance function when the rectangles are still overlapping.

5. Conclusion

5.1 Contributions and conclusions

This thesis focused on developping an algorithm which is calculating the optimal distribution of the rooms in a flat. As indicated in the introduction the idea of writing an algorithm to place computer chips on the mother board, was jointly with architechtural specialists transfered to optimizing space management in construction works on physical buildings.
Calculating the different options of sorting the rooms needs a high capacity of calculation. Because a normal solver would not be sufficient to calculate nonlinear issues, we used IPOPT, an interior point optimizier. Using IPOPT required selected programming languages to write the algorithm. Here we wrote the codes in C++, so it was adaptable to IPOPT.

In a first part we are refering to some other works which are already done and treat similar issues.
To get the reader familiar with the methods we explained them briefly in the second chapter. This was followed by the description of the algorithms and then by the results of the work.

Calculating the overlap of two rooms was the first task and could be done in a few code lines. Writing the algorithm to calculate the gradient was more complicated. The gradient of the function is not steady, so IPOPT calculates it during the process repetitively. So it was necessary to include all different possibilties which could be calculated. After several attempts and revisions, the gradient

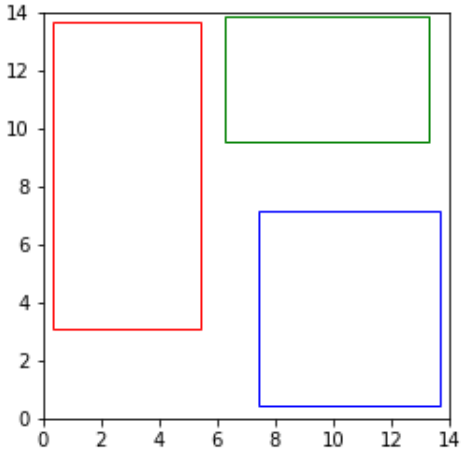


Figure 9: Plot of the results by running the algorithm with only the overlap function and no additional one.

was calculated correctly.

Because calculating the overlap is not always sufficient, we implemented a function to maximize the distance between the centers. This function helps in some cases, but nevertheless it is not working in all constellations.

The idea of an artificial architect is certainly a way to be further developped and will be asked for in the future, in order to digitalize everything.
To conclude, it should be possible to write an algorithm which pays attention to all different kinds of constraints of how a flat or how the rooms in a flat should best be located. However, because of its complexity the given variables, the considerable amount of additional work and the inavailability of existing literature, this project should be investigated and further pursued in a continuing work.

5.2 Personal Evaluation

During my degree and my master studies I acquired a profound mathematical knowledge. I learned to program in python, R and a tiny bit in C++. Because my degree was practical oriented I have chosen more practical topics in my master, too. This brought me to decide writing my thesis about this issue.

I increased my C++ skills imense during this process. Further through reading many articles and the support of my advisor I professionalized my knowledge in using algorithmic functions to calculate specific cases.

This work gave my a deep view in how research on an academic environment is made and how difficult it can be, to do research in these complex fields. It learned me how time intensive it is, trying to write new algorithms or calculation methods, as well as the understanding of correlations among different subjects and areas .

For my personal progression I know that it is important to keep an overview and not to lose myself in specific details. I learned that if I got stuck somewhere, it could be more efficient to try to get help from experts or just outsiders having a different view on the topic. Because of the global situation of COVID-19 and the quarantine, I have experienced that scientific exchange is a new challenge when meetings from person-to-person are impossible. Doirect communication is a really important part of such a work and meeting in person and discuss the issues are certainly more efficient than the online communication.
Regular meetings and exchanges with peers, managed by a senior person could help avoiding unnecessary efforts and structure an efficient procedure. Obviously, this years events were not supportive in that sense.

5.3 Future work

To continue the work one can optimize the code that it will eliminate the overlap in every case, not depending on the inital values. If this is working, one could try to implement constraints, which are defining a corridor or doors, so that every room is reachable. Furthermore, one could include the numerous additional variables, such as windows, water pipes, electronical access etc.

6. Bibliography

In the bibliography you will find the references and the list of figures.

References

[1] Chen, Mao and Huang, Wenqi *A two-level search algorithm for 2D rectangular packing problem.* ScienceDirect, 2006.

[2] Martinez, Thierry and Vitorino, Lumadaiara and Fages, Francois *On Solving Mixed Shapes Packing Problems by Continuous Optimization with the CMA Evolution Strategy.* arXiv, 2019.

[3] Demiröz, Baris Evrim and Altinel, I. Kuban and Akarun, Lale *Rectangle Blanket Problem: Binary integer linear programming formulation and solution algorithms.* arXiv, 2019.

[4] Wächter, Andreas *IPOPT documentation*
`https://coin-or.github.io/Ipopt/index.html`

[5] Wikipedia contributors *Computational geometry*
`https://en.wikipedia.org/w/index.php?title=Computational_geometry&oldid=962665297`

[6] Wikipedia contributors *Gradient descent*
`https://en.wikipedia.org/w/index.php?title=Gradient_descent&oldid=962649577`

[7] Wikipedia contributors *IPOPT*
`https://en.wikipedia.org/w/index.php?title=IPOPT&oldid=939146110`

List of Figures

Confidentiality statement for academic works (bachelor's and master's theses, etc.)

Confidentiality statement

Mr Jordi Cortadella, the supervisor, coordinator or tutor of the academic work entitled Mathematical Optimization For Architectural Layout Design deposited by the student Moritz Otth, declares that:

( ) the academic work **contains confidential information** (according to the conditions detailed below)

(x) the academic work **does not contain confidential information**

Confidentiality period and justification

[Fill in this section only if you have declared that the work contains confidential information.]

The undersigned declares that the confidentiality of the academic work must be maintained for the period of time given below:

( ) **until** _____

(x) confidentiality must be observed **indefinitely**

The undersigned declares that the reasons for this confidentiality are the following:

( ) an evaluation of the possibility of protecting the work is pending

( ) a third party has expressed an interest in marketing the work

( ) it is part of a larger piece of research that is subject to a confidentiality agreement with a company

( ) other_____

Public dissemination of the work

[Fill in this section only if you have declared that the work contains confidential information.]

The undersigned authorises the following type of dissemination of the work in the institutional repository UPCommons or any other platform that may replace it:

(X) dissemination of the **full text of the work** starting on the date indicated in the previous section (if the author has authorised this dissemination)

( ) dissemination of the work's **bibliographic details** (not the full text)

( ) **no dissemination** because the work contains confidential information (pursuant to Article 37.1 of Spain's Intellectual Property Law, the UPC's Libraries, Publications and Archives Service shall file the work in UPCommons and shall not permit public access to the text or to the corresponding bibliographic details, thus ensuring its confidentiality, preservation and conservation)

Barcelona, 28.09.2020

The supervisor, coordinator or tutor:

[x] I consent to the UPC processing the personal data collected in this form in accordance with this notice on information and access to personal data:

Information and access to personal data	
Identity and contact details of the controller	Universitat Politècnica de Catalunya. Servei de Biblioteques, Publicacions i Arxius C/Jordi Girona, 31 (Building K2M, Planta S1, Office *S103-S104, Ca*mpus Nord). 08034 Barcelona Telephone*: +34 93 401 58 28* info.biblioteques@upc.edu
Contact details of the data protection officer	Universitat Politècnica de Catalunya proteccio.dades@upc.edu More information here: https://www.upc.edu/normatives/ca/proteccio-de-dades/normativa-europea-de-proteccio-de-dades/dades-de-contacte-del-delegat-de-proteccio-de-dades
Purposes of the processing	*[*F05.8.*]* Deposit of academic production in the Institutional Repository. In case of doubt, contact with: proteccio.dades@upc.edu
Legitimate interests	Consent. More information here: https://www.upc.edu/normatives/ca/proteccio-de-dades/normativa-europea-de-proteccio-de-dades/legitimacio/view
Recipients or categories of recipients	With your explicit consent, your contact information will be communicated to the person interested in your work for educational, research or promotion purposes
Rights	Right of access by the data subject. Right to rectification or erasure ('right to be forgotten'). Right to restriction of processing. Right to object. Right to data portability. More information here: https://www.upc.edu/normatives/ca/proteccio-de-dades/normativa-europea-de-proteccio-de-dades/drets
Period for which the personal data will be stored	As needed for any of the purposes that are described in our retention policy. More information here: https://www.upc.edu/normatives/ca/proteccio-de-dades/normativa-europea-de-proteccio-de-dades/politica-de-conservacio-de-les-dades-de-caracter-personal
Complaint	If you have been unable to exercise your rights to your satisfaction, you can file a complaint with the APDCAT. apdcat.gencat.cat